平成20年度　　卒業研究論文

# 匿名通信路のノード離脱に対する通信路継続方式の提案

指導教官
齋藤 彰一 准教授

名古屋工業大学　情報工学科
平成16年度入学　16115011番

石黒 聖久

# 目 次

# 第1章

# はじめに

近年、インターネットの普及により、重要な情報の通信をインターネットで行う機会が増えている。これらの情報の通信には厳重な情報守秘が必要となる。現在、暗号化技術の発展により、それらのデータの守秘は実現されているが、通信を行っているという事実そのものを守秘することに関しては完全とは言えない。しかし、電子投票や内部告発等、匿名性を必要とした通信は多く、これからも増加していくと思われる。一般的に通信における匿名性とは次の三つを指す。

- 送信者の匿名性
- 受信者の匿名性
- 送信者、受信者の関係性の匿名性

現在、代表的な匿名通信手法の一つに OnionRouting[1] がある。OnionRouting では複数の中継ノードを介し、データの多重暗号化を行うことで、上記三つの匿名性を満たした通信を行うことが出来る。しかしこの手法には、送信時の経路上にある中継ノードが 1 つでもその位置を離れたとき、返信データが送信者に届かず、復旧が困難という欠点を伴う。

本論文ではこの OnionRouting をベースに、切断時の経路を修復を目的とした、ノードの管理を行う別の層を追加することで、より一般的な使用に耐えうる匿名通信方式を提唱する。

以下、二章では本研究の基盤となっている関連研究について述べ、三章では本研究の提案手法の解説を行う。四章では三章で述べた提案手法の実装について述べる。五章では評価結果とその考察、六章で結論をまとめる。

# 第2章

# 関連研究

本章では多重暗号化により匿名通信を行う OnionRouting[1] と、ノード管理を行うための通信の際に利用する Chord[2] について説明を行う。

## 2.1 OnionRouting

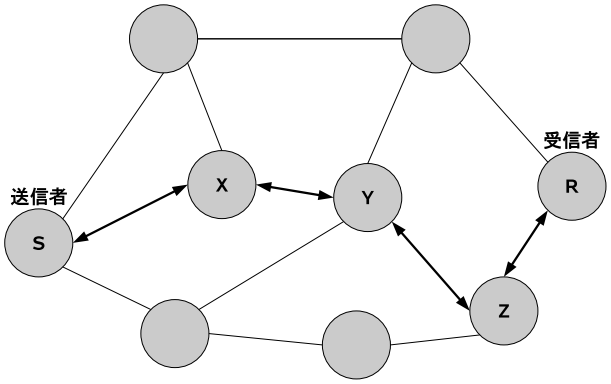


図 2.1: OnionRouitng 全体図

OnionRouting[1] は、ネットワーク上に複数配置された中継ノードで構成される。送信者が受信者に対してデータを送信する際、送信するデータを中継ノードの公開鍵を用いて多重に暗号化する。各中継ノードは受信した暗号メッセージを復号し、次ノードの情報を得る。これを受信者までの全中継ノードで繰り返し行う。

例として、図2.1の通信網において、送信者Sから受信者Rに対してメッセージを送信する場合を説明する。最初に送信者は経由する中継ノードを決定し、中継するノード及び受信者の公開鍵を取得する。また、中継ノードをX、Y、Zとし、それぞれのノード及び受信者に対応する公開鍵を$Key_X$、$Key_Y$、$Key_Z$、$Key_R$とする。通信内容をVとし、それぞれのノードのアドレスを$N_X$、$N_Y$、$N_Z$、$N_R$とする。またα ∥ βを要素αとβを結合したものとする。

最初に受信者が受信すべきデータVを、受信者の公開鍵を用いて暗号化する。この暗号化されたデータを$M_R$=$Key_R$(V)とする。続いて$M_R$と受信者のアドレス$N_R$を結合した上で、受信者の一つ手前の中継ノードであるノードXの公開鍵で暗号化したデータ$M_X$=$KeyX$( $N_R$ ∥ $M_R$)を作成する。以下同様に、データ$M_X$とノードXのアドレス$N_X$をノードYの公開鍵$Key_Y$で暗号化してデータ$M_Y$の作成。データ$M_Y$とアドレス$N_Y$をZの公開鍵で暗号化する。

これによって下記のメッセージMが完成する。

$$\mathrm{M}=Key_X(\ N_Y \parallel Key_Y(\ N_Z \parallel Key_Z\ (\ N_R \parallel Key_R\ (\mathrm{V}))))$$

送信者Sは、メッセージMを最初の中継ノードXに対して送信する。メッセージを受け取ったノードXは、自身の秘密鍵を用いてメッセージの復号を行う。復号に成功すると、ノードXは次の中継先ノードYのアドレス$N_Y$と、送信するデータ$M_Y$を取得することが出来る。以下同様にノードXはノードYに$M_Y$を送信。Yはメッセージを受信し復号する。取得したデータ$M_Z$をノードZへ送信。これの繰り返しによって受信者Rは$M_R$を受信する。

以上の中継処理を踏むことによって、送信者、受信者が中継ノードに紛れ込み、匿名性を得ることが出来る。また中継するノードは自身の前後の経路上ノードの位置しか把握出来無いため、中継ノードが一つでも信頼出来れば、匿名性は維持される。

### 2.1.1 問題点

OnionRoutingの問題点として、所持する経路情報が少ないために経路維持が困難という欠点がある。例えば図2.1の例において、ノードYが離脱した場合を考える。ノー

ドYの隣接ノードであるノードXとZはその離脱を検知することは出来る。しかし、ノードXの持つノード情報はノードSとYのみであり、ノードZも同様にノードYとRの位置情報しか持たない。そのためノードYが離脱してしまう事で、経路の維持が不可能となってしまう。このように、基本的には中継ノードの離脱による経路の切断に対応することが出来ない。

## 2.2 Chord

Chord[2] はハッシュを用いてノードを管理する手法の一つである。P2P ネットワークにおいて、サーバを用いること無く高速にコンテンツの検索、ルーティングを行う。

Chord では ID 番号 0〜$2^m$-1 による環状ノード群を形成する。各ノードは Chord に参加する際に、自身のコンテンツをハッシュに掛け ID を取得し、その ID 番号のノードとなる。各参加ノードは FingerTable と呼ばれる経路表を管理する。FingerTable には自身のノード ID+$2^i$mod($2^m$-1) の ID を担当するノードとの接続情報を保持しており、コンテンツの検索の際に用いられる。ただし Successor(自ノードの ID よりも大きな ID を担当するノードの中で最小の ID を担当するノード) と、Predecessor(自ノードの ID よりも小さな ID を担当するノードの中で最大の ID を担当するノード) という両隣接ノードとは、常に接続を保持している。またノードの離脱に備えた Successor List と呼ばれるリストを保持する。Successor List には、自身の Successor 回りの数ノード分の接続情報を持つ。これらの List や Table は定期的な接続の確認を行い、必要なら修復を行う。

コンテンツの検索時には自身の持つ FingerTable を用いる。目的のコンテンツをハッシュに掛け、求められた Key の値以下で最も大きい値の ID 番号を担当するノードへ検索要求を出す。検索要求を受けたノードも同じように検索を行う。これを繰り返すことで目的のコンテンツを持つノードの検索が完了する。

図 2.2 に Chord の検索例を示す。あるノード (N10) があるキー (Key:65) を探索する場合、N10 は自分が保持している FingerTable の中から key:65 に一番近いノード (N45) に対して問い合わせを送る。N45 が key:65 を管理していない場合、同様に自分が保持

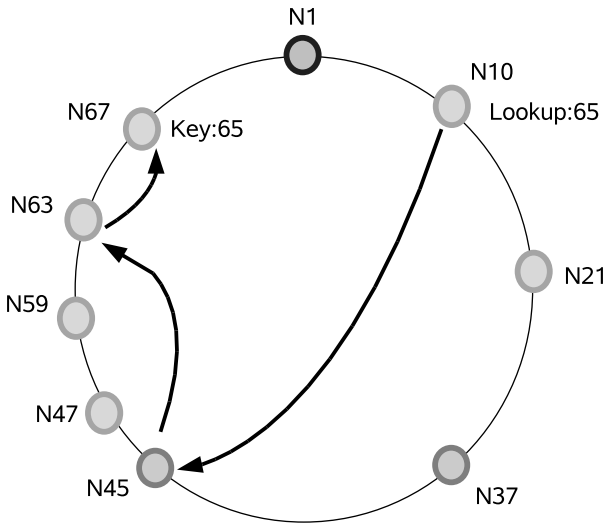


図 2.2: Chord による検索

している FingerTable の中から key:65 に一番近いノード (N45) に対して問い合わせを送る。このようにして検索キーを管理するノードにリクエストを送り、コンテンツを得る。このように、N ノードの中から目的のノードの発見する問い合わせの回数は, 最悪でも O(logN) 回となる。また P2P における分散ネットワークでは、ノードが離脱する場合には、持っているコンテンツごと離脱するため、以後そのコンテンツを参照できなくなる問題がある。Chord ではその問題に対して, あらかじめコンテンツのレプリカを Successor に渡しておくことで解決をしている。

# 第3章

# 提案手法

本論文では従来手法であるOnionRoutingを拡張し、ノードの離脱による匿名通信路切断に対応可能な通信路を提案する。OnionRoutingが切断に対応するのが困難であった理由は、匿名性の関係上、一つのノードが所持する経路情報が制限されていたことによる。一つのノードが多くの他のノード情報を持つと匿名性が成り立たたせるのが困難になる。そこで匿名通信を行う匿名通信路層とノードの管理を行うノード管理層とで機能を分離することで、それぞれの機能の持つノード情報の因果関係を絶つ。例えば、ノードAがノード管理層でノードB,C,Dの情報を管理している時に、匿名通信路で自分の前後以外の中継ノードがノードB,C,Dであっても、ノード管理層でのB,C,Dだと認識することが出来ない。これにより匿名性を維持しつつ、ノード間の管理を行うことが可能となる。ノード管理層におけるネットワークにはChordを用いる。

## 3.1 全体構成

提案手法の構成を図3.1に示す。ノードは起動時に公開鍵、秘密鍵のペアを作成して、鍵サーバへ登録する。その後、ノード管理層のChordによる環状ネットワークに参加する。ノード管理層はノードの参加と離脱、匿名通信路の切断に備えた経路情報の保管を行う。匿名通信路層はノード管理層の上位に位置し、匿名通信に用いるノードを決定し、経路の構築と暗号化による実際の通信を行う。

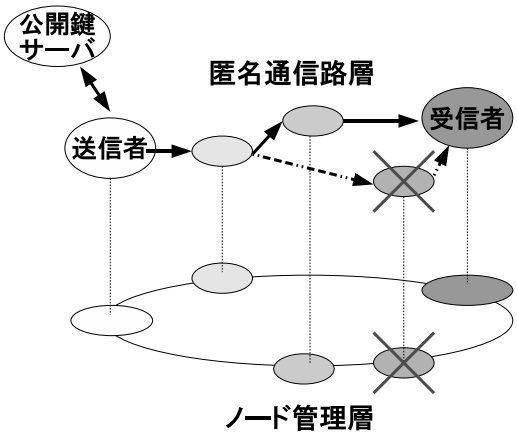


図 3.1: 全体構成

## 3.2 匿名通信路層

匿名通信路層では、匿名通信の経路決定と構築を行い、暗号化処理を施した上で実際の通信を行う。匿名通信路には OnionRouting を用いる。以降ではメッセージの最初の送信ノードを Sender、中継を行うノードを Router、最終的なメッセージの受信ノードを Receiver という。また、通信路における Receiver 側から見た i 番目のノード（Sender が n 番目）を $N_i$ とする。以下同様に i 番目のノードの公開鍵と秘密鍵をそれぞれ $PubKey_i$ と $PriKey_i$ といい、経路生成時に i 番目のノードが受信するメッセージを $M_i$ という。この通信路では、初回の経路生成メッセージの生成には公開鍵と秘密鍵を用いたメッセージを作成するが、以後の通常通信におけるノード間の通信には、暗号復号に掛かる時間の削減のために共通鍵を用いる。i 番目のノードと j 番目のノード (i<j) の通信に用いる共通鍵を $Key_{i,j}$ とする。以上を表 3.1 に記す。

表 3.1: 記号一覧

| | |
|---|---|
| ノード | $N_i$ |
| 共通鍵 | $Key_{i,j}$ |
| 公開鍵 | $PubKey_i$ |
| 秘密鍵 | $PriKey_i$ |
| メッセージ | $M_i$ |
| ノードの持つ情報 | $Info_i$ |
| Succesor に渡す情報 | $S\_HalfKeys_i$ |
| Predessesor に渡す情報 | $P\_HalfKeys_i$ |

表 3.2: $Info_i$ の内容

| | | |
|---|---|---|
| 位置情報 | $N_{i+1}$ | $N_{i-1}$ |
| 共通鍵 | $Key_{i,i+1}$ | $Key_{i-1,i}$ |
| 秘密鍵 | $PriKey_i$ | |

匿名通信路での動作は以下の三種類に分かれる。以下、それぞれについて述べる。

- 匿名通信路生成時
- 匿名通信時
- 匿名通信路切断時

### 3.2.1 匿名通信路生成時

以下、送信者 Sender、中継者 Rotuer と受信者 Receiver の処理について述べる。Rotuer と Receiver に関しては途中までの処理が同じであるため、まとめて説明を行う。

#### Sender

Sender は通信の際に経路生成メッセージの作成を行う。メッセージの生成の際に、多重暗号化に用いる公開鍵とそのノードの位置情報を、鍵サーバから取得する。中継するをランダムで決定し、続いてそれらのノードの共通鍵を生成する。メッセージの生成手順は 2 章で説明した通りとなるが、図 3.2 のように $N_i$ と $M_i$ 以外に共通鍵 $Key_{i,i+1}$、$Key_{i-1,i}$ を含んだメッセージとなる。

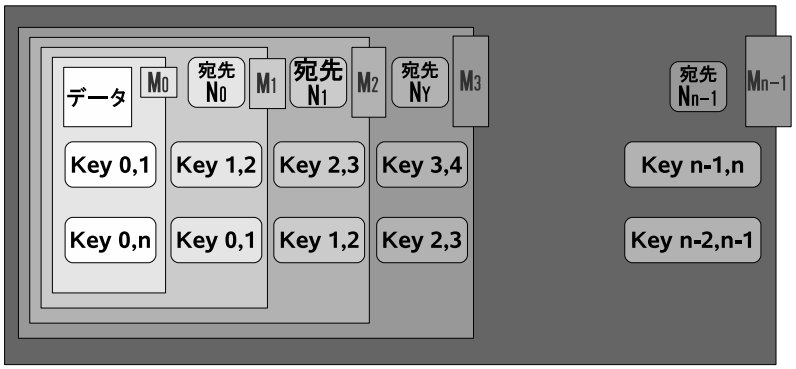


図 3.2: 経路生成メッセージ 2

**Router と Receiver**

メッセージ $M_i$ を受け取ったノードは、$PriKey_i$ を用いて復号を行う。復号した際に自身が Router か Receiver かの判定がされる。Router であれば、復号したメッセージ $M_i$ に含まれる共通鍵 $Key_{i,i+1}$ と $Key_{i-1,i}$ を取得し、メッセージ $M_{i-1}$ を次の経由先ノード $N_{i-1}$ へ送信する。Receiver であれば、同じように共通鍵 $Key_{0,1}$ と $Key_{0,n}$ を取得し、経路生成完了のメッセージを生成した通信路を用いて Sender へ送信する。

### 3.2.2 匿名通信時

Sender は通信を行う際に、送信するメッセージを $Key_{0,n}$ で暗号化した上で、$Key_{0,1}$ で更に暗号化し、$N_1$ へ送信する。Router であるノード $N_i$ は、メッセージを受信後に $Key_{i,i-1}$ で復号したメッセージを、$Key_{i,i-1}$ で暗号化して $N_{i-1}$ へ送信する。Receiver は受信したメッセージを $Key_{0,1}$ で復号し、更に $Key_{0,n}$ で復号することで、Sender の送信したメッセージを取得することが出来る。

### 3.2.3 ノード離脱時

ノード $N_i$ の離脱時、離脱したノードの前後に位置する $N_{i-1}$ と $N_{i+1}$ が離脱を検知する。検知したら、ノード管理層によって選出された $N_i$ の代理ノードからの接続がある

まで待機する。代理ノードからの接続があれば、$Key_{i-1,i}$,$Key_{i,i+1}$ を用いて、代理ノードの認証を行った上で通信の再開を行う。代理ノードからの接続によって通信が回復したら、離脱前と同様に通信を行う。

## 3.3 ノード管理層

ノード管理層ではノードの離脱に備えたデータのコピーと、ノードの離脱による匿名通信路の切断の復帰を行う。ノード管理層でのノード間通信には Chord を用いる。Chord は全てのノードの情報を持つこと無しに、任意のノードの検索を行えるため、匿名性の観点から有利であるために採用した。

ノード管理層は、ノードの起動時と同時に Chord の環状経路に参加し、機能し始める。ノード管理層での動作は以下の三種類に分かれる。以下、それぞれの動作について述べる。

- 代理ノード選出
- ノード離脱時
- 匿名通進路破棄時

### 3.3.1 代理ノードの選出

ノード $N_i$ がメッセージ $M_i$ を受け取った際に、ノード管理層ではノードの離脱による匿名通信路切断に備えるために、代理ノードの選出を行う。代理ノードには $N_i$ の Successor が選出される。この時、選出された代理ノードには離脱に備えたデータ $Info_i$（表 3.2 を参照）のコピーを渡しておく必要がある。それらのデータは安全性確保のために二つに分割し、図 3.3 の様に、代理ノードとなる Successor に $S\_HalfInfoi$ を、Predeccesor に $P\_HalfInfoi$ を保管する。

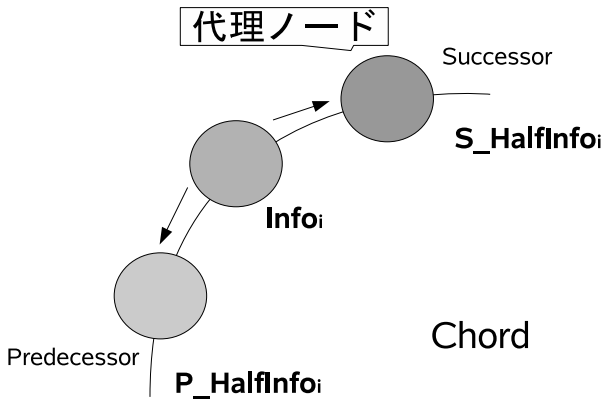


図 3.3: 代理ノードの選出

### 3.3.2 ノード離脱時の処理

ノードが離脱した時、Chord のアルゴリズムによりノード管理層の環状ネットワークが修復される。修復後に離脱したノードの Succesor は Predessesor に $P\_HalfInfoi$ を要求する。$S\_HalfKeyi$ と受け取った $P\_HalfKeyi$ を再構築し、$Info_i$ を取得した後に、Succesor は離脱したノードの役割を引継ぐ。役割を引き継いだ代理ノードは、離脱したノード $N_i$ が担当していた位置の通信を行う。通信を始めるに当たっては、$N_{i-1}$,$N_{i+1}$ に接続要求を出し、$Key_{i-1,i}$,$Key_{i,i+1}$ を用いて認証作業を行う。認証が完了したら通信を再開する。

### 3.3.3 匿名通信路破棄時

経路破棄時には、中継を行っていたノード $N_1$〜$N_{n-1}$ は、それぞれの Succesor と Predessesor に保管した $Info_i$ の破棄を指示する。破棄を確認次第、自身の持つ $Info_i$ を破棄する。

# 第4章

# 実装

実装に当たっての前提として、公開鍵及び位置情報を保管する鍵サーバとの通信は保証されているとする。また本研究では Sender、Receiver の離脱には対応しない。

## 4.1 OverlayWeaver

OverlayWeaver[3] とは Overlay 構築ツールの一つである。これを用いることで、オーバレイ設計者はたかだか数百ステップで structured オーバレイのアルゴリズムを実装することができる。計算機 1 台の上で繰り返し試験を行うことが出来き、どのアルゴリズム実装を使うかを実行時に決めることができる。またエミュレータを用いて数万ノードの実験を行うことができるので、設計者は新規、既存アルゴリズム間の比較を大規模かつ公正に行うことも出来る。今回ノード管理層に用いる Chord がサポートされているため使用する。

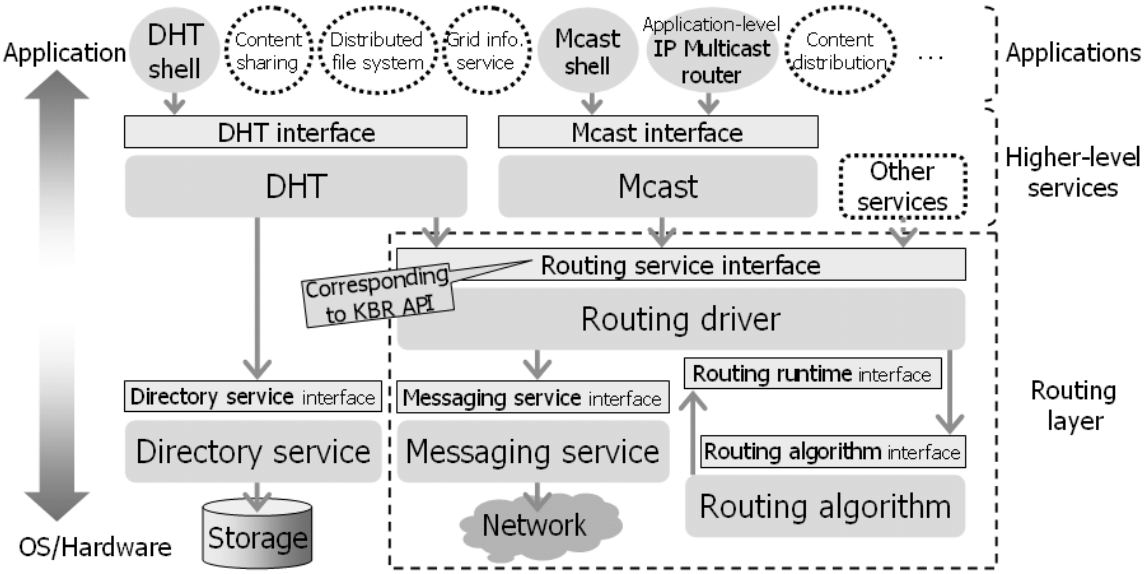


図 4.1: OverlayWeaver

OverlayWeaver には様々な機能が実装されているが、今回は以下の機能のみを利用する。Routin Driver には iterative ルーティングを、Routing Algorithm には Chord を用いる。また、秘密鍵、公開鍵には RSA2048bit を、共通鍵には AES128bit を用いる。匿名通信路間の通信プロトコルには TCP を用いる。

- DHTshell、DHTInterface、DHT
- Routing Service Interface、Routing Driver、Routing Runtime
- Routing Algorithm Interface、Routing Algorithm
- Messaging Service

提案システムは図 4.1 における Routing driver の部分に組み込む。公開鍵の登録等は、起動時の OverlayWeaver の初期化と共に行われる。図 4.1 における DHTShell に匿名通信開始コマンドを追加し、通信の実際の手続きは隠蔽してある。匿名通信路の生成時に Sender から送信される経路生成メッセージ、及び Info を分割した $P\_HalfInfo$、$S\_HalfInfo$ の送受信には、図 4.1 における Messaging service を用いる。

## 4.2 全体の流れ

全体の流れを図 4.2 に示す。最初の OverlayWeaver 起動時に公開鍵、秘密鍵を生成し、鍵サーバへ登録を行う (図 4.2:Step 0)。以後、ユーザからの匿名通信開始コマンドの入力または匿名通信路生成メッセージを受信するまで待機する。

匿名通信開始コマンドの入力を受けたノードは、匿名通信生成メッセージを生成する (図 4.2:Step 1)。生成したメッセージは、Messaging service を用いて Router となるべきノードに送信し、自身は以後匿名通信路層においては Sender として機能する。匿名通信路生成メッセージを受信したノードは、自身の秘密鍵を用いてメッセージを確認し、そのメッセージの中継者か受信者かを判断する (図 4.2:Step 2)。中継者であれば匿名通信路層では Router となり、そのメッセージを次のノードに送信し、代理ノードの選出処理を行う (図 4.2:Step 4)。受信者であれば匿名通信路層では Receiver となり、通信路生成完了のメッセージを Sender に対して送信する。匿名通信には OverlayWeaver の機能とは切り離された匿名通信路層の通信路を用いて通信を行う (図 4.2:Step 3)。

ノードが離脱に伴う匿名通信路の切断時、(図 4.2:Step 4) で選出された代理ノードは経路修復処理を行う (図 4.2:Step 5)。以降はそれぞれの Step の詳細説明に入る。

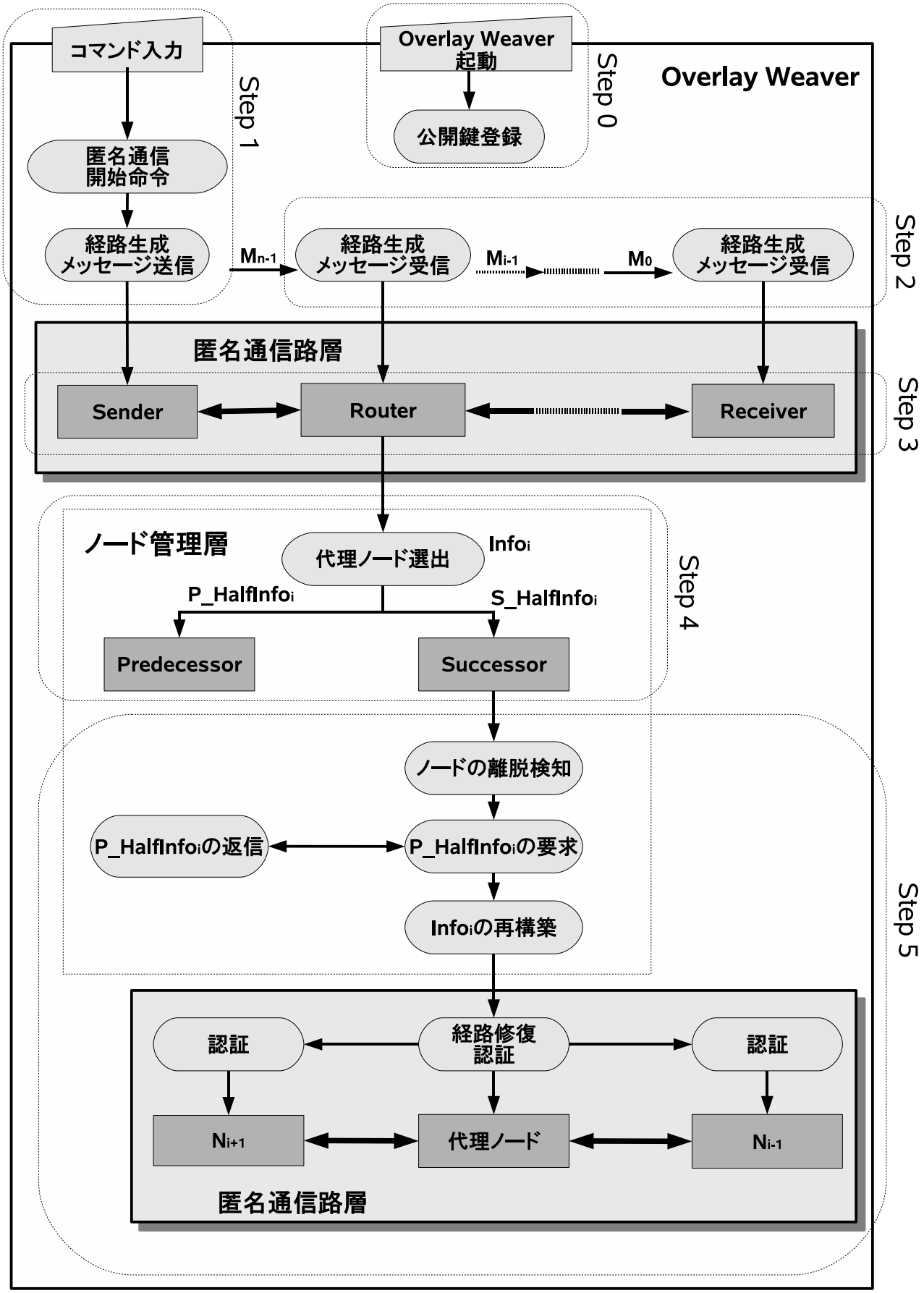


図 4.2: OverlayWeaver と提案手法モデル

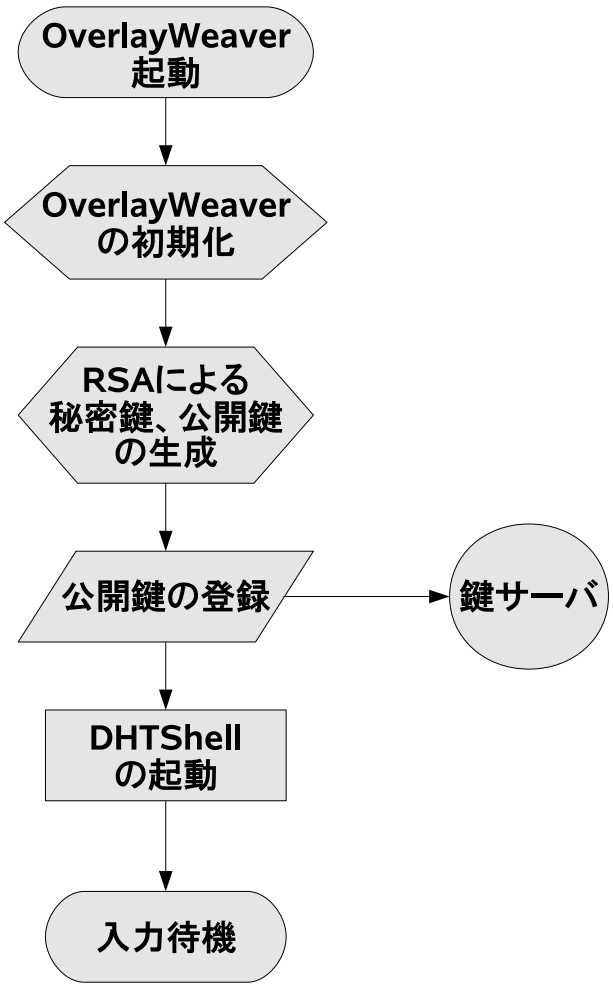


図 4.3: OverlayWeaver 起動時

## 4.3 Step0: OverlayWewaver 起動時

この Step の一連の流れを図 4.3 に示す。組み込んだ機能は、DHT の初期化が完了した後に初期化する。起動時に秘密鍵、公開鍵を作成し公開鍵および自身の位置情報を鍵サーバへ登録する。その後、DHTShell による匿名通信開始コマンドの入力を待機する。

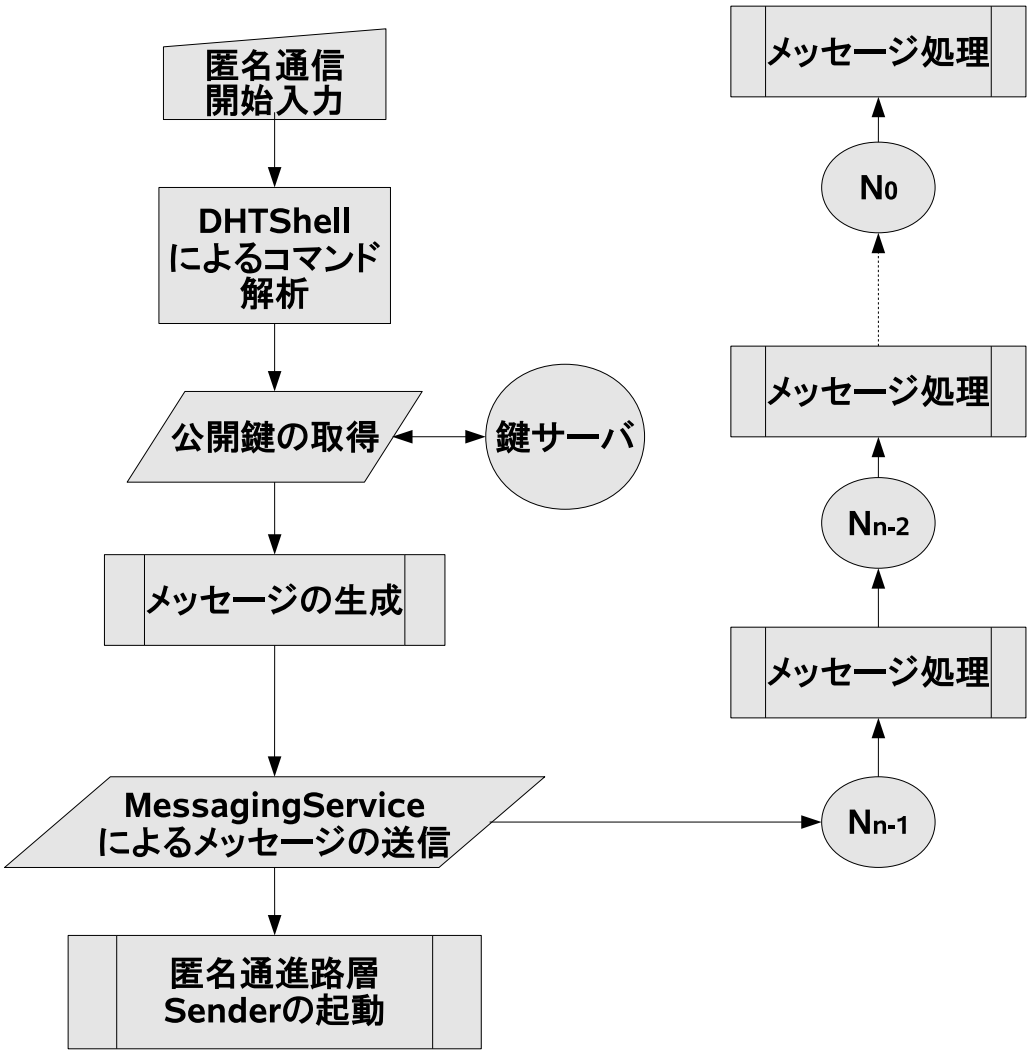


図 4.4: 匿名通信路生成時 (Sender)

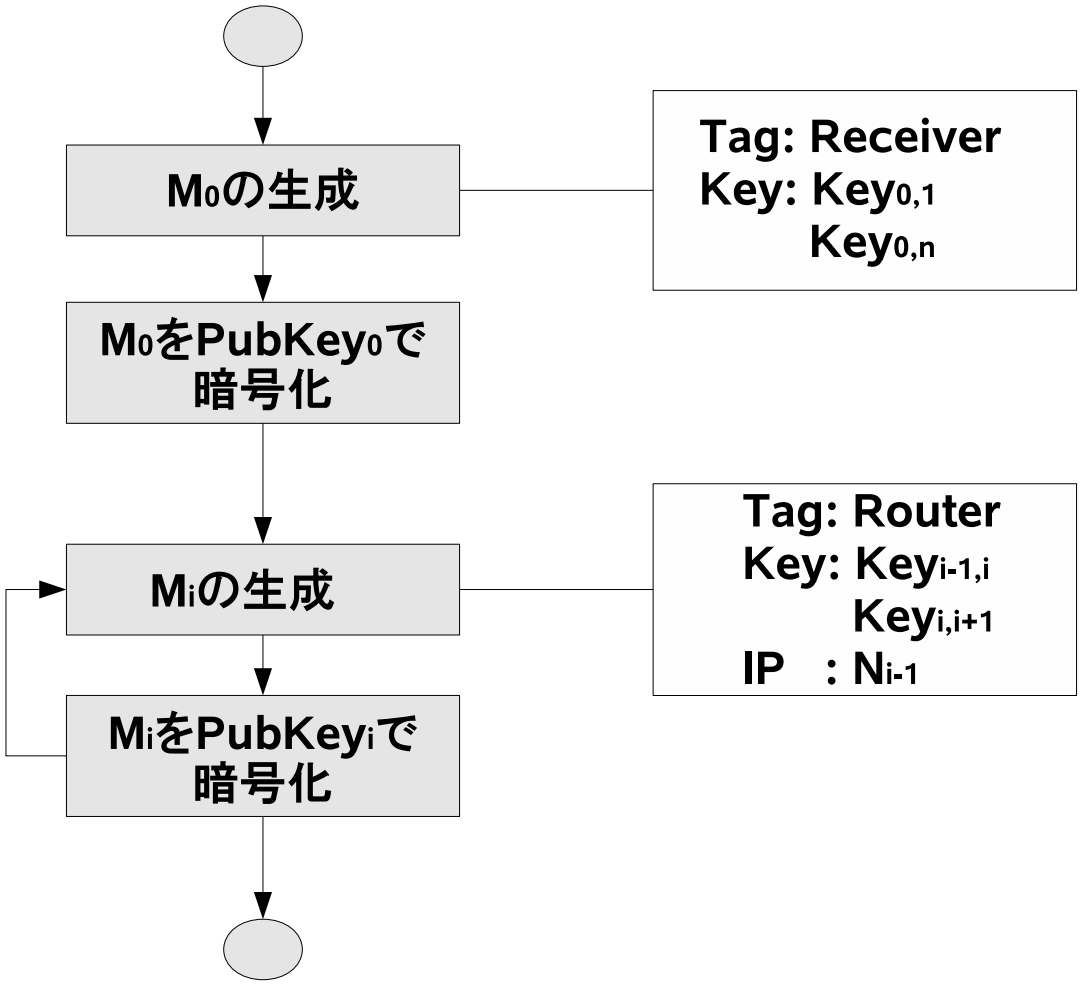


図 4.5: 匿名通信路生成メッセージ作成

## 4.4 Step1: 匿名通信路生成メッセージの作成時

この Step の一連の流れを図 4.4 に示す。匿名通信路層の生成は、DHTShell からの匿名通信開始コマンドを入力されたノードから行われる。このノードが Sender となり、匿名通信路生成メッセージを受信したノードが Router または Receiver となる。

### 4.4.1 Sender

ユーザは Receiver となるノードを指定して鍵サーバへ公開鍵の要求をする。Sender は経由するノードの位置情報と、そのノードの公開鍵を取得する。経由するノード数はランダムで決まる。公開鍵を受け取った後に経路生成メッセージの生成を開始する。

メッセージ生成の流れを図 4.5 に示す。最初に Receiver が受信、復号するメッセージ $M_0$ を作成する。このメッセージには Sender との通信用共通鍵 $Key_{0,n}$ と、Router$N_1$ との通信用共通鍵 $Key_{0,1}$ 含む。このメッセージを $PubKey_0$ を用いて暗号化する。なお共通鍵には AES による 128bit 共通鍵を用いる。以降 $M_1$ から $M_{n-1}$ までの構成は同じである。Router が受信、復号するメッセージ $M_i$ には $M_{i-1}$ と $key_{i,i+1}$、$key_{i-1,i}$、及び $N_{i-1}$ の位置情報が含まれている。このメッセージを Messaging Service を用いて $N_{n-1}$ へ送信する。送信した後は 4.6.1 の動作に移る。

## 4.5 Step2: 匿名通信路生成メッセージ受付処理

この Step の一連の流れを図 4.6 に示す。経路生成メッセージ $M_i$ を受け取ったノードはメッセージを自身の秘密鍵 $PriKey_i$ を用いて復号する。メッセージ内容により、そのノードが Router または Receiver となる。

### 4.5.1 Router

自身が中継ノードであれば、メッセージ $M_i$ を受け取ったノードは以後 Router として振る舞う。Router はメッセージ $M_i$ に含まれる共通鍵 $Key_{i,i+1}$ と $Key_{i-1,i}$ を取り、メッセージ $M_{i-1}$ を次の経由先ノード $N_{i-1}$ へ Messaging Service を用いて送信する。送

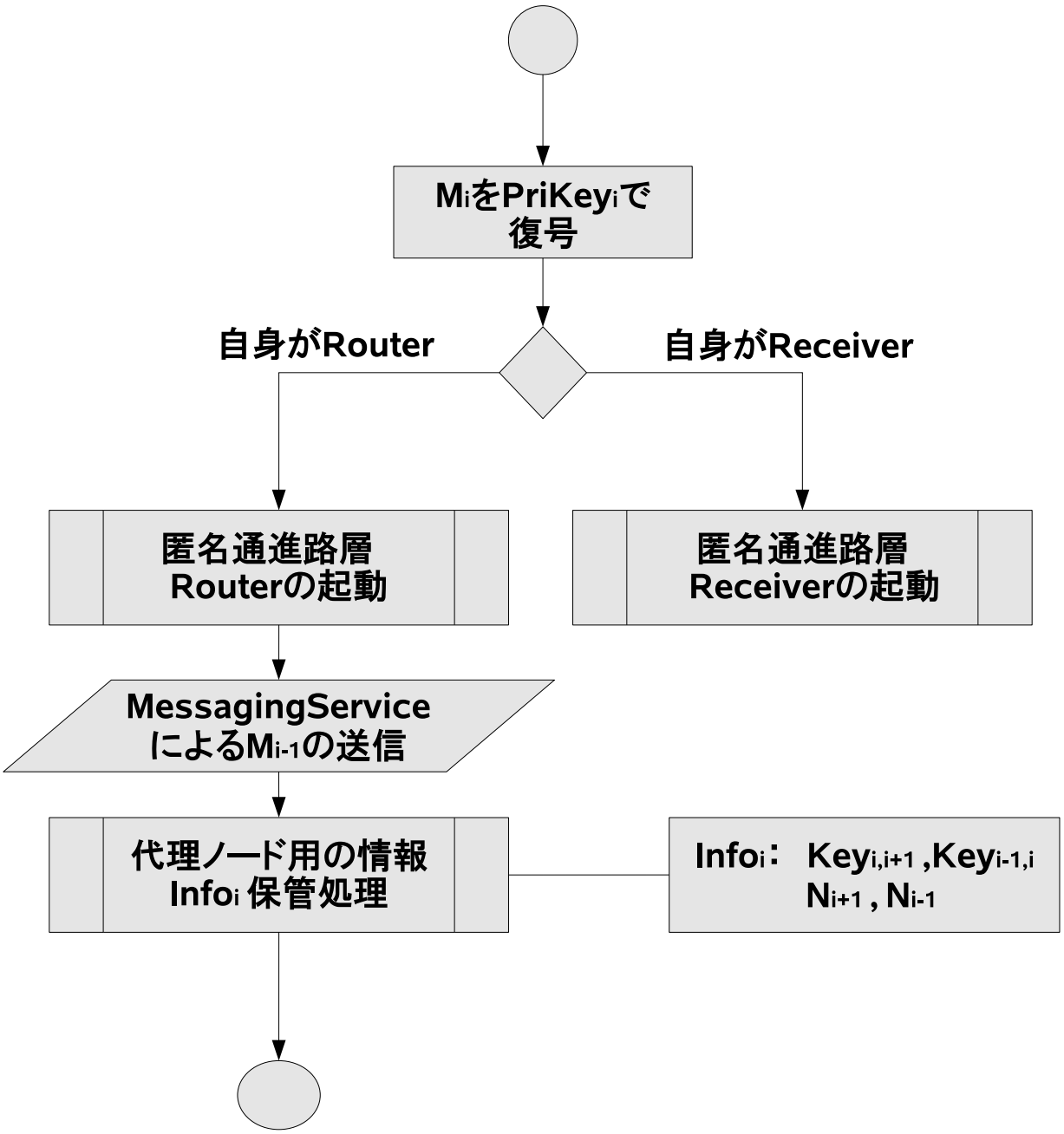


図 4.6: 匿名通信路生成メッセージ受付処理

信した後は図、4.6.3 の動作に入る。

### 4.5.2 Receiver

メッセージが自分宛であれば、メッセージ $M_i$ を受け取ったノードは以後 Receiver として振る舞う。Receiver はメッセージ $M_0$ に含まれる共通鍵 $Key_{0,1}$ と $Key_{0,n}$ を取る。送信した後は図 4.6.4 の動作に入る。

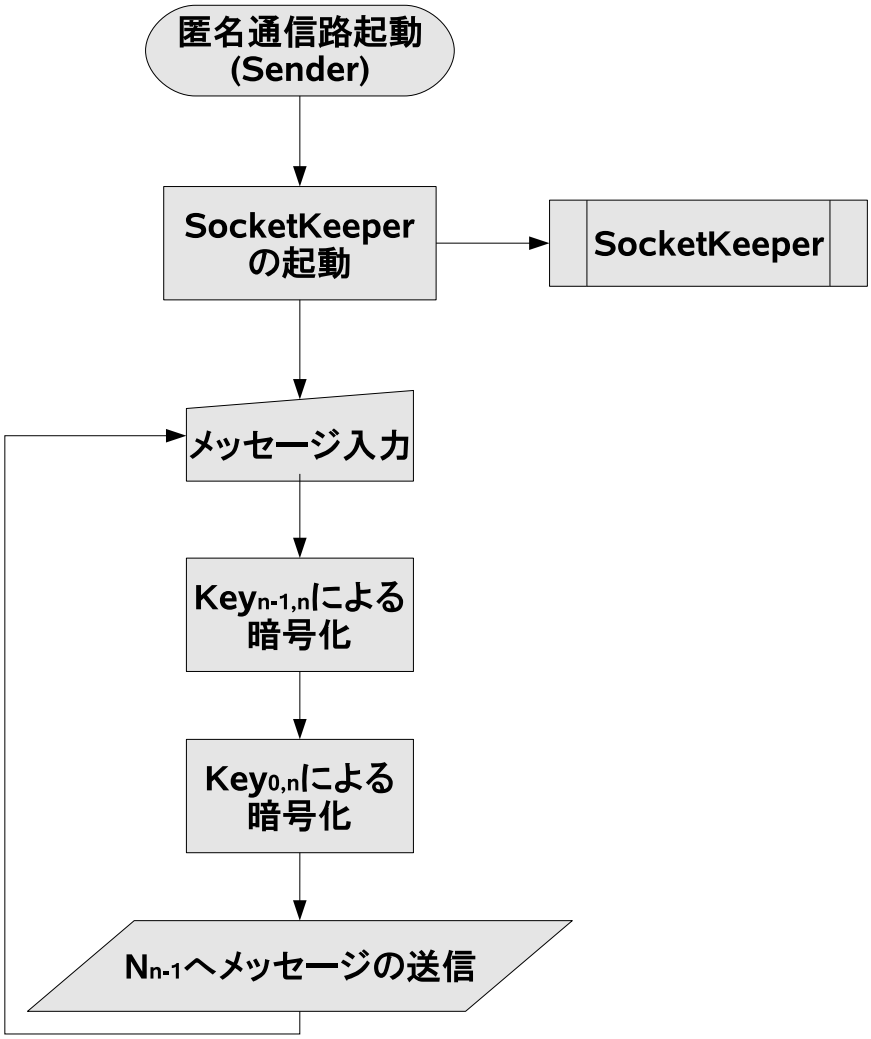


図 4.7: 匿名通信 (Sender)

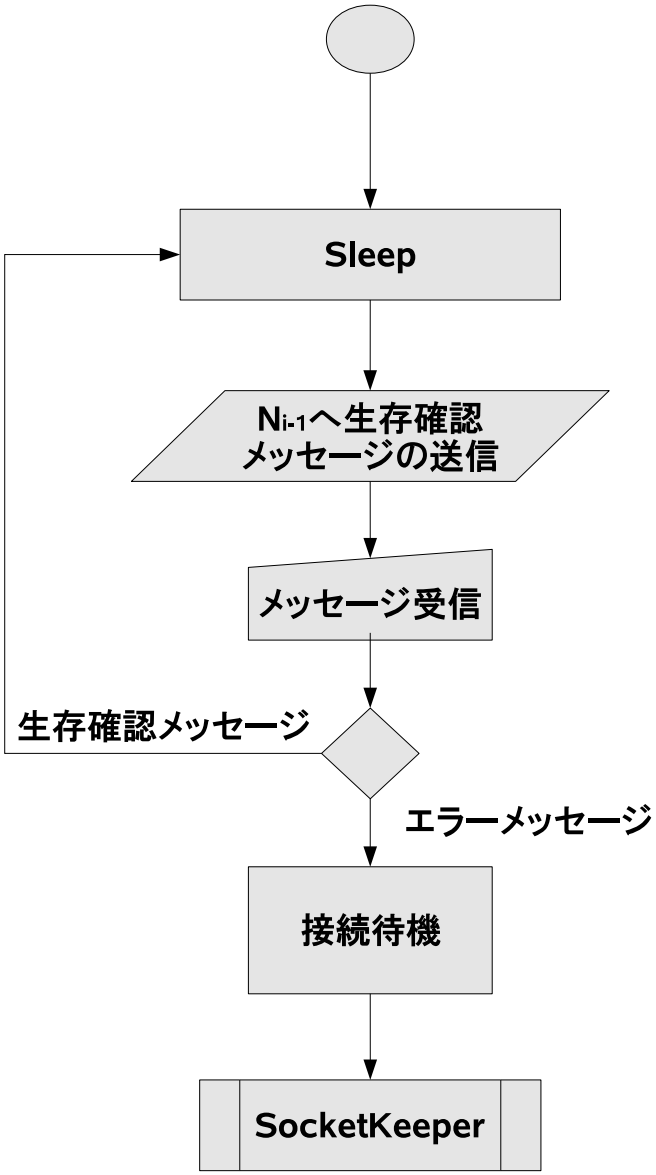
Sleep
Ni-1へ生存確認
メッセージの送信
メッセージ受信
生存確認メッセージ
エラーメッセージ
接続待機
SocketKeeper

図 4.8: 通信路監視

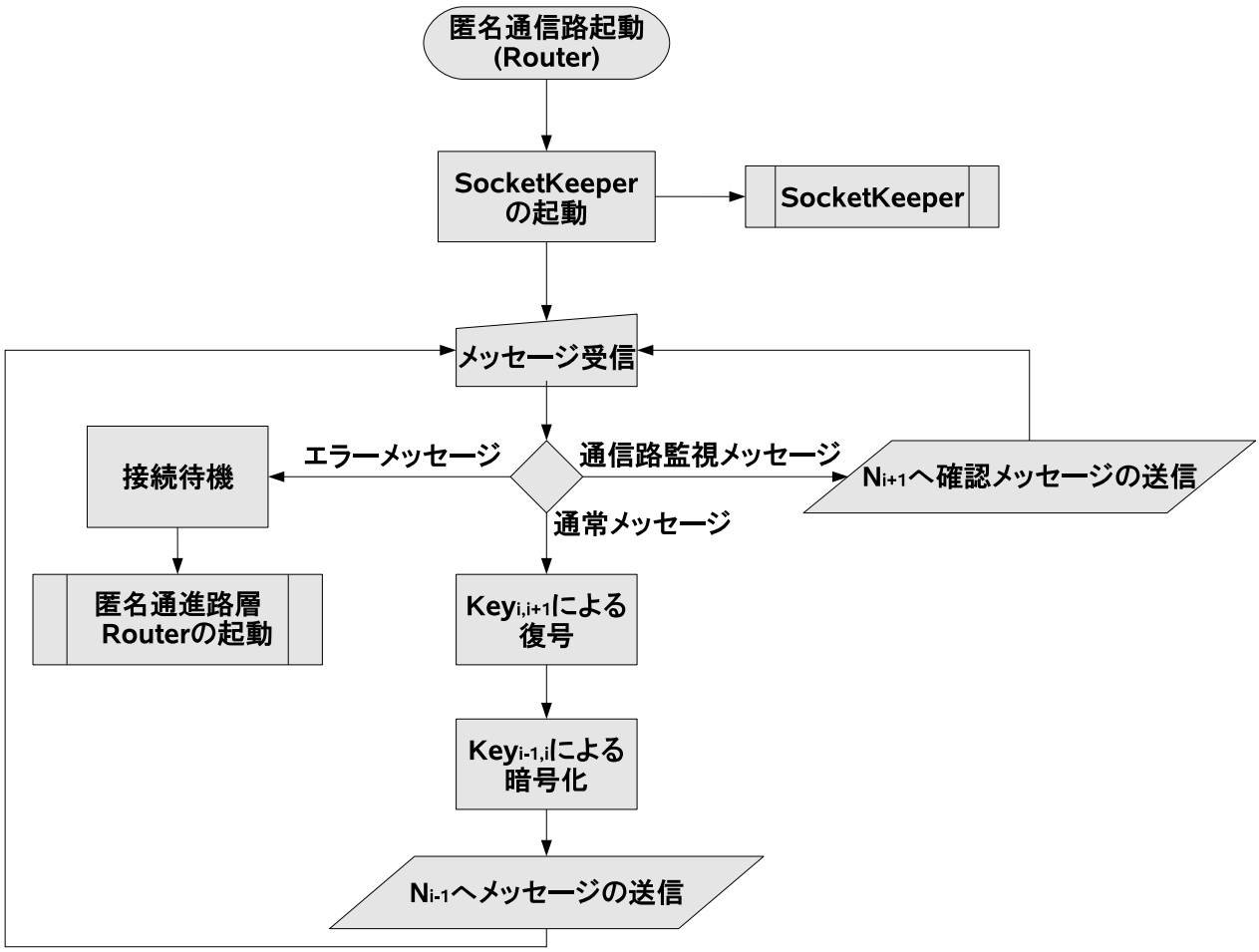


図 4.9: 匿名通信 (Router)

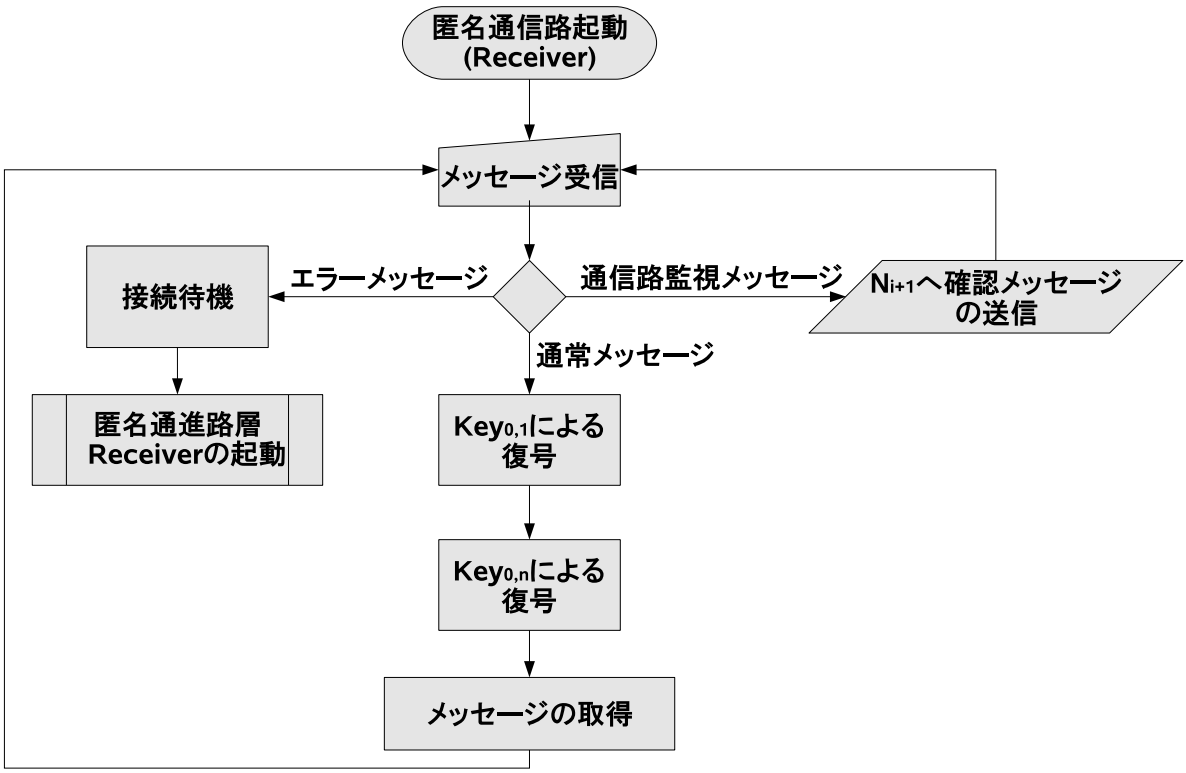


図 4.10: 匿名通信 (Receiver)

## 4.6 Step3: 匿名通信時

匿名通信は OverlayWeaver とは切り離された匿名通信路層の通信路を用いて行う。匿名通信を行うノードの動きは、メッセージの送信者である Sender、メッセージの中継者である Router、メッセージの受信者である Receiver のそれぞれに分かれる。

### 4.6.1 匿名通信 (Sender)

Sender の一連の流れを図 4.7 に示す。Sender は通信の監視を行うスレッド 4.6.2 を建てた後、通常のメッセージを送信する動作に入る。Sender は入力されたメッセージを $Key_{0,n}$ で暗号化した上で、さらに $Key_{n-1,n}$ で暗号化した通常メッセージを $N_{n-1}$ に送信する。

### 4.6.2 通信路監視スレッド

通信路監視スレッドの一連の流れを図 4.8 に示す。Sender と Router では、通信とは逆方向のノードの生存の確認を行うために、定期的にメッセージの送信を行うスレッドを建て、監視する。通信路監視メッセージを送信後、受信したメッセージが生存確認メッセージかエラーメッセージかを判断する。

- 生存確認メッセージ

  生存確認メッセージであれば、ノードは機能していると判断する。一定時間後、再度通信路監視メッセージの送信を行う。

- エラーメッセージ

  エラーメッセージであれば、ノードは離脱したものと判断する。離脱したノードの代理ノードからの接続を待機し、接続されたら再び監視スレッドを建てなおす。

### 4.6.3 匿名通信 (Router)

Router の一連の流れを図 4.9 に示す。Router は通信の監視スレッド図 4.6.2 を建てた後、メッセージの中継をする動作に入る。Router はメッセージを受信した際に、そのメッセージが通常メッセージ、通信路監視メッセージ、エラーメッセージかを判断する。

- 通常メッセージ

  通常メッセージであれば、メッセージを $Key_{i+1,i}$ で復号した後に $Key_{i,i-1}$ で暗号化し、$N_{i,i-1}$ に送信する。

- 通信路監視メッセージ

  通信路監視メッセージであれば、生存確認メッセージを返信する。

- エラーメッセージ

  エラーメッセージであれば、代理ノードからの接続を待機し、接続されたら再び匿名通信 (Router) 図 4.9 を再起動する。

### 4.6.4 匿名通信 (Receiver)

Receiver の一連の流れを図 4.10 に示す。Receiver はメッセージを受信した際に、そのメッセージが通常メッセージ、通信路監視メッセージ、エラーメッセージかを判断する。

- 通常メッセージ

  通常メッセージであれば、$Key_{0,1}$ で復号した後に $Key_{0,n}$ で復号することでメッセージを取得する。

- 通信路監視メッセージ

  通信路監視メッセージであれば、生存確認メッセージを返信する。

- エラーメッセージ

  エラーメッセージであれば、代理ノードからの接続を待機し、接続されたら再び匿名通信 (Receiver) 図 4.10 を再起動する。

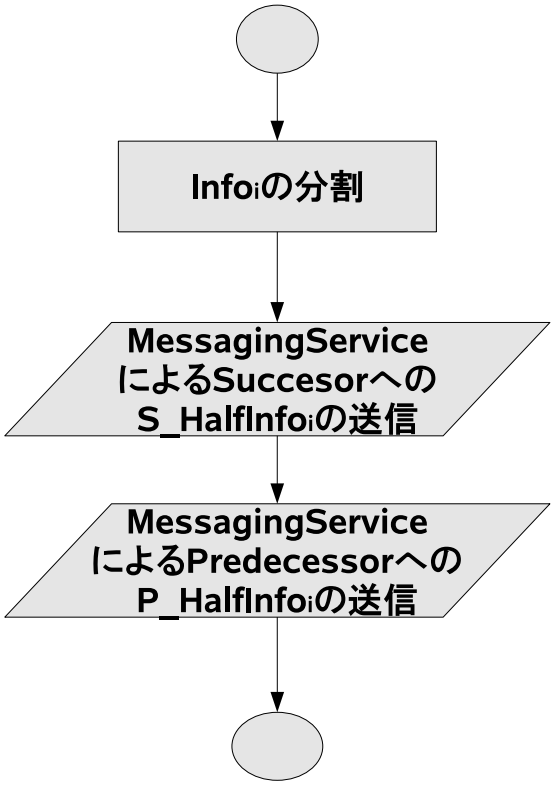


図 4.11: 経路情報分割処理

## 4.7 Step4: 代理ノード選出時

代理ノードの選出時の一連の流れを図 4.11 に示す。図 4.6 の MessagingService でノードが経路生成メッセージを受け取った際、自身が Rotuer であれば自身の代理となるノードを選出し、自身の持つ匿名通信路維持に必要な情報 3.2(以後 $Info_i$) を byte 単位で分割し、Messaging Service を通じて、Predessesor と Successer に預ける。以後 Predessecor

に預けた情報を $HalfKeys_{Predessecor}$、Succesor に預けた情報を $HalfKeys_{Succecor}$ とする。

## 4.8 Step5: 離脱時の処理

ノード離脱時の処理の一連の流れを 4.12 に示す。ノード離脱が起きた時、Routing Algolizm では Finger Table、Successor List の更新、及び経路の修復が行われる。この時、離脱したノードが匿名通信路に参加しているか否かを判定し、離脱によって切断された匿名通信路の修復も行われる。

ノードの離脱を検知した際、自身が Successor であれば、Messaging Service によって Predessesor に $HalfKeys_{Predessecor}$ の要求を出す。要求を受けた Predessesor は $HalfKeys_{Predessecor}$ を Successor に返す。この時、離脱したノードの Predessesor は引き続き、離脱したノードの Succesor に対する Predessesor となるため、$HalfKeys_{Predessecor}$ は引き続き保管する。

Successor は受け取った $HalfKeys_{Predessecor}$ と $HalfKeys_{Succecor}$ を再構築し、離脱したノードの持っていた情報を復元する。復元が完了したら、匿名通信路における離脱したノードの前後のノードに接続要求を出す。接続後、$Key_{i-1,i}$ と $Key_{i,i+1}$ を用いてノードの認証作業を行う。認証が完了したら代理ノードは Router4.9 となり、通信が再開される。

## 4.9 経路破棄時の処理

経路破棄時には Router は RoutingService を用いて、Predessesor と Successer に $HalfKeys_{Predessec}$ と $HalfKeys_{Succecor}$ の破棄を指示するメッセージを送信する。

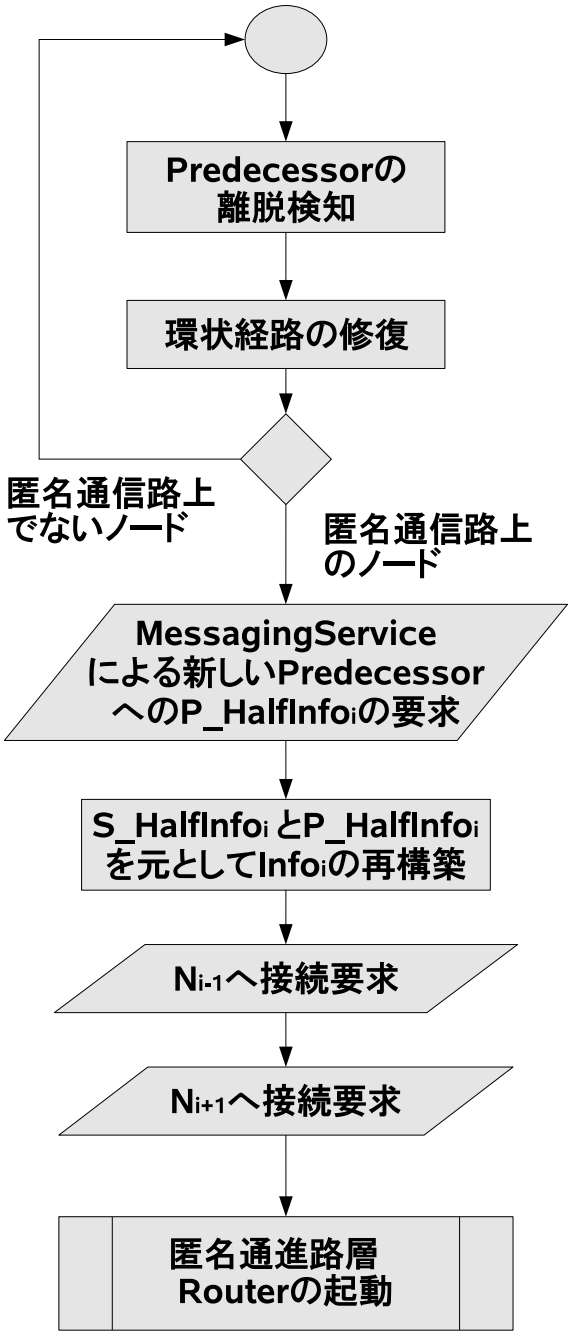
Predecessorの
離脱検知
環状経路の修復
匿名通信路上
でないノード
匿名通信路上
のノード
MessagingService
による新しいPredecessor
へのP_HalfInfoiの要求
S_HalfInfoi とP_HalfInfoi
を元としてInfoiの再構築
Ni-1へ接続要求
Ni+1へ接続要求
匿名通進路層
Routerの起動

図 4.12: 離脱時の処理

# 第5章

# 評価

実験評価はノードの離脱による匿名通信路の切断から、その経路の復旧にかかる遅延の計測を行う。

## 5.1 実験環境

実験は全て計算機上で行った。計算機環境は以下の通りである。

CPU Pentium4 / 3.4GHz
Memory 2.0Gbyte
OS VineLinux

OverlayWeaver Ver.0.8.9
Eclipse Ver3.4.0

今回は一台の計算機上で複数のコンソールから同時に起動することにより、仮想的に四台の計算機としてシュミレーションを行った。全ノード数は４つで、ノードの離脱にはOverlayWeaverのExitコマンドを中継ノードの内の一台で実行することで、ノードとしての機能を停止させることで行った。以下の計測時間は、ノードの離脱から経路の復旧が完了するまでに要した時間である。全部で十回行い、そのうちの最大、最小、平均遅延を算出した。

表 5.1: 測定結果

| | |
|---|---|
| 最小遅延 | 58(s) |
| 最大遅延 | 117(s) |
| 平均遅延 | 90(s) |

## 5.2 結果と考察

測定結果を表 5.1 に示す。

これらの値はネットワーク遅延を考慮していない状態での値であるため、実際には更に時間が掛かると思われる。最小遅延が約 1 分、最大遅延が約 2 分、平均遅延が約 1 分 30 秒となった。現時点で復旧にこれだけの時間が掛かるのは現実的でない。しかし、これらの時間の大半は OverlayWeaver での、ノードの離脱からその検知に要する時間である。これは代理ノードの起動が、離脱の検知を行った Predessecor と Succecor の、Successor List と Predessecor List の更新を完了した後に行われるからである。これに対する今後の課題としては、本システムの実装を OverlayWeaver のより Chord Algorizm に近い層でノード離脱検知を行うことで、ノードの離脱による匿名通信路の切断から、復旧に掛かる時間の短縮を行うことが挙げられる。

修復機能を実装したことによる問題として、送信者、受信者の離脱には対応していないことが挙げられる。そのためこれらが離脱すると、「代理ノードが起動しない」ことによって、送受信者の直前のノードにそれを推定されてしまう可能性がある。

またノードの情報を他のノードに渡しているために、鍵を預けられた Succesor と Predessesor が結託した場合、Router 間の通信を傍受される恐れがある。しかし、データ内容自体が暗号化されているため、傍受してもデータを盗み見られることはない。

特殊なケースとして、離脱したノード $N_i$ の Successor や Predessesor が、匿名通信路における $N_{i-1}$ または $N_{i+1}$ と一致する場合がある。この時、$N_{i-1}$ または $N_{i+1}$ が $N_i$ の代理ノードとなってしまい、経由すべきノードの数が減ってしまう。これケースが中継ノードの最大数と同じ回数分だけ起こると、送受信者が特定される。しかし頻繁に起こることでは無いため、今回は想定外とした。仮りに想定する場合でも、鍵を分割保管する際に確認することで回避可能である。

# 第6章

# まとめ

本研究では匿名通信路における、ノードの離脱による経路の切断からの復旧する手法を提案した。提案手法では匿名通信路を行う通信路の他に、ノードの管理を行う通信路を用意することで、それぞれの通信路での因果関係を無くす。匿名通信路とノード管理用通信路を別層にすることによって、匿名性を大きく損なうこと無く、匿名通信路全体を把握することが可能となる。

匿名通信路には、強固な匿名性を持つ OnionRouting を用いる。ノード管理用通信路には、少ないノード情報で任意のノードを検索可能な Chord を用いた。実装に当たっては、実装の難しい Chord をサポートしている OverlayWeaver を用いた。

今後の課題としては経路切断からの復旧に掛かる時間の短縮、公開鍵の管理を行う鍵サーバの存在、送信者の前後のノードが離脱することによる匿名性の低下の改善等が挙げられる。

# 参考文献

[1] Onion Routing and Online Anonymity Matt Hooks and Jadrian Miles April 30, 2006

[2] Stoica, I. , Morris, R. , Karger, D. , Kaashoek, F. and Balakrishnan, H. : Chord; A Scalable Peer–To–Peer Lookup Service for Internet Applications, Proc. 2001 ACM SIGCOMM Conference, pp. 149–160(2001).

[3] OverlayWeaver http://overlayweaver.sourceforge.net/index-j.html

# 謝辞

本研究のために多大な御尽力を頂き、日頃から熱心な御指導を賜った名古屋工業大学の齋藤彰一准教授に深く感謝致します。

また、本研究の際に多くの助言、協力をして頂いた松尾啓志教授、津邑公暁准教授、松井俊浩助教、及び齋藤研究室ならびに松尾・津邑研究室の皆様に深く感謝致します。